An Integrated, but not Exact-Representation, Editor/Formatter

Richard Furuta
*University of Maryland**

ABSTRACT
Integrated Editor/Formatters merge the document editing and formatting functions into a unified, interactive system. A common type of Integrated Editor/Formatter, the Exact-representation Editor/Formatter (also known as WYSIWYG), presents an interactive representation of the document that is identical to the printed document. Another powerful metaphor applied to documents has been to describe the document as abstract objects—to describe the document's logical structure, not its physical makeup. The goal of the research reported here is to merge the flexibility found in the abstract object-oriented approach with the naturalness of document manipulation provided by the Exact-representation Editor/Formatters. A tree-based model of documents that allows a variety of document objects as leaves (e.g., text, tables, and mathematical equations) has been defined. I suggest a template-oriented mechanism for manipulating the document and have implemented a prototype that illustrates the mechanism. Further work has concentrated on handling user operations on arbitrary, contiguous portions of the display.

1. Motivation and Goals of the Research

The world of text formatters can be divided into two parts.[1] In one group are the *pure formatters*, which convert a document description, prepared by a separate editing system, into a formatted document suitable for display on an appropriate hardware device. In the other group are the *Integrated Editor/Formatters*, which merge the editing and the formatting functions into one unified, interactive system—documents are created, viewed, and revised without leaving the editor/formatter.

*This work was carried out at the University of Washington. This paper is a summary and condensation of [Fur86]. I thank Alan Shaw and Pierre MacKay for their advice and assistance.

[1] We have given more complete definitions of this terminology in [FSS82].

A powerful metaphor applied in the pure formatter domain has been to view the document as consisting of abstract objects [Rei80, Uni84, Lam85]. The system's user describes the document's logical structure instead of its physical makeup, and therefore can concentrate on specifying the document's content. The document description can be used more flexibly since the description of the document's contents and structure is separated from definition of its appearance. Consistent, wholesale modification of the appearance of the document's objects can be made without change to the document's description. Similarly, the document's text can be used in different contexts without requiring change.

A powerful metaphor applied in the Integrated Editor/Formatter domain has been to present the interactive representation of the document as identical to the paper representation, maintaining all the fonts, sizes, and spacings in both forms [Lam78, SIKV82, App84]. I call an editor/formatter of this type an *Exact-representation Editor/Formatter.* Much of the popularity of such editor/formatters is due to the naturalness of the editing process—the representations and manipulations that take place are similar to those that are used with paper documents. A negative factor, however, is the inflexibility of the description—others have called these editor/formatters "What you see is what you get" systems. Brian Kernighan's well known retort was to describe them instead as "What you see is *all* you've got" systems.

The overall goal of my research is the design of an Integrated Editor/Formatter that merges the naturalness of document manipulation provided by an Exact-representation Editor/Formatters with the flexibility inherent in an abstract object representation. To achieve this, the abstract objects and their semantic relationships must be displayed in a way consistent with their concrete representation on paper and manipulation of the abstract objects should be carried out through manipulation of this concrete representation. As a consequence, the design avoids interactive operations that operate on hidden data structures or in which the target of an operation is not directly connected to the context in which the operation is issued. This rules out, for example, a system that shows a paper-like representation of the document on the display but in which the operations are traditional tree-traversal commands, phrased in terms of an underlying (and invisible) tree structure.

2. The Model of Documents

A document consists of abstract objects, hierarchically related to each other. A wide variety of leaf object types are desirable and necessary

at the lowest level of the hierarchy; for example, objects containing textual material, tabular material, mathematical formulæ, line drawings, or scanned images (bits). Additionally, these lowest-level objects may interact with each other—for example, tabular objects may contain textual or mathematical objects within the individual entries of the table.

A tree structure provides an attractive approximation of the object relationships. However, the tree representation cannot adequately model the wide variety of structures found at the leaves. A tree is not powerful enough in some cases: tables are not naturally tree structured because the entries in the table have multiple parents (row and column). In other cases, a tree representation is awkward because it is *too* powerful: paragraph text is most naturally treated as a string of characters.

For this reason, a hybrid structure, the **tnt**, is used to model the document.

The highest level structure of a **tnt** is an ordered tree. The relationships between objects in the tree can be and are represented and constrained by a context-free grammar. The objects found in this higher level structure are said to be in the *strict tree* portion of the **tnt**.

At the "leaves" of the strict tree portion are instances of the various *tree blocks*. An arbitrary structure is found within a tree block—the only requirement is that it be possible to define a representation-independent ordering between the components located within the tree block. The structures within the tree blocks collectively make up the *free structure* portion of the **tnt**.

There is exactly one kind of structure in the strict tree portion of the **tnt** but there may be many different kinds of structures in the free structure portion. The terminating nodes of any particular free structure may be defined to be either *atoms* or *transition nodes* (or both). Atoms are actual terminating points in the **tnt**. A transition node is a terminating point for the structure that contains it and also a root for an enclosed strict tree structure. Thus, a **tnt** is built from alternating structures—first the strict tree, then a tree block enclosing a free structure, and then perhaps a transition node that leads to another strict tree. This alternation is reflected in the term "**tnt**," which stands for "strict tree—not strict tree."[2]

[2]All objects are ordered with respect to each other in a **tnt**. When the relationships between two document objects cannot be constrained in this manner, as is the case with floating figures and footnotes, the objects are placed in separate **tnts**, which are related to each other by a linking scheme. [Fur86] discusses the forest of **tnts** in more detail. The discussion in this paper is limited to the processing of a single **tnt**.

Looking at the **tnt** from the bottom up, it can be seen that the basic building blocks of the document are the tree blocks. These tree blocks are related to one another through the hierarchical relationships expressed by the structure of the strict tree. These hierarchical relationships differ between document classes and are what distinguish different document classes, for example the document class that defines a document divided into sections and the document class that describes a business letter. The *same* tree blocks can be used in each of the different document classes.

My document model is derived from one defined by Shaw [Sha80, FSS82].[3] Other researchers have independently developed similar document models. A partial list includes Borkin and Prager's work [BP80], Peels, et al.'s COBATEF [PJN85], and Quint and Vatton's GRIF [QV86].

2.1. Internal and External Representations

The **tnt** is the *internal representation* of the document. A system that processes a document does so by mapping between the internal representation and one or more external representations. Each mapping may be either unidirectional or bidirectional.

External representations are put to many uses in a document processing system. An external representation that is modified interactively presents the document to the user of the system and allows modifications to be made to the document. An interactive representation could be highly oriented to the appearance that the document would take if it were to be printed on paper, in which case the interface probably would resemble that of the Exact-representation systems, or the representation could be highly oriented to manipulation of abstract objects.

Also of use are external representations that present condensed or symbolic versions of the document, for example, the document in outline (or table of contents) form or the document as a picture of a tree. When the summary representation is bidirectional, manipulations on it can be used to alter and restructure large parts of the document or can be used to move about quickly in the document.

File-oriented external representations are also desirable. For example in this work, a bidirectional external representation, called the "storage representation," is provided to allow documents to be saved and restored

[3]Another document model, also based on Shaw's work, has been defined by Kimura and Shaw [KS84, Kim84]. In Kimura and Shaw's model, the document is represented by a single homogeneous structure. In my model, the representation of the document is through a heterogeneous collection of structures.

```
<document>       = article-title authors (<paragraph>|<section>)+
<section>        = section-id section-title (<paragraph>|<section>)+
<paragraph>      = [text-block|tbl-block|eqn-block|<itemized-list>|
                       <enum-list>|<quotation>]+
<itemized-list> = <i-item>+
<i-item>         = item-marker <paragraph>+
<enum-list>      = <e-item>+
<e-item>         = enum-marker <paragraph>+
<quotation>      = <paragraph>+

<tbl-entry>      = <paragraph>+
```

Figure 1: The grammar describing a sectioned document

across sessions. A useful unidirectional class of file-oriented representations are the "formatter representations," which serve as *input* to a batch-oriented formatter such as TEX, Scribe, or **troff**.

The relaxation of the constraints imposed by exact-representation allows the specification of different external representations for specific purposes. For example, a interactive representation need not incorporate pagination, thereby simplifying its design considerably. However, a paginated representation of the document still can be obtained through a formatter representation.

3. A Prototype System

A central part of this work has been the design and development of a prototype system, sketching the means for manipulating the **tnt**. Three external representations are included in the prototype: a storage representation, a formatter representation,[4] and an interactive representation.

The interactive representation will be the topic of the remainder of this paper.

3.1. The Grammar and Augmentations

The syntax of the context-free grammar that defines the **tnt** is slightly modified from the BNF to allow the prototype's interactive representation to be driven directly from the grammar.

A terminal in the grammar is further classified as being either user-defined terminal or system-defined terminal, respectively representing

[4]The formatter representation is driven by translation tables and the table to produce LATEX input has been designed and is being tested.

information supplied by the system's user and information generated by the system. In the grammar of Figure 1, for example, `text-block` is a user-defined terminal representing a block of text and *section-id* is a system-defined terminal representing the number associated with a document's section.

Each nonterminal must appear on the left hand side of exactly one grammar production. The right hand side of the production rule consists of a sequence of *components* where each component is either a grammar terminal, a grammar nonterminal or a *group*. A group is a set of grammar elements, where each element is either a grammar terminal or grammar nonterminal. When the production is expanded, exactly one of the elements contained within the group will be present in the expansion. Syntactically the group is bracketed and the elements of the group are separated from one another with the "|" symbol.

It has been convenient to provide two categories of groups in the grammar description, called *choose one* and *choose all*. The distinction was made to help direct the way in which the prototype interactive representation displays the unexpanded elements from the group. Choose one is used when the user of the system is expected in general to choose only one of the alternatives presented within the group. Choose all is used when the user of the system is expected to want to use all the alternatives presented within the group.

Choose one groups are syntactically identified in a production by being enclosed in square brackets. Choose all groups are enclosed parentheses.

Each of the components of the right hand side may be single (present in the expansion only once), starred (present zero or more times), or plussed (present one or more times). Note that the elements of a group cannot be starred or plussed.

3.2. The Prototypical User Interface

The intention in the prototypical user interface is not to provide an "exact representation" of the document as it would appear on paper but a "sufficient representation" that gives a good intuitive feeling for what the elements are of the document but that does not necessarily indicate what the exact details of the placement of these elements will be in other representations.

The approach taken in the prototypical user interface is to provide a template-driven system for creation and modification of a document, directed by and constrained by the grammar defined above. Manipulation

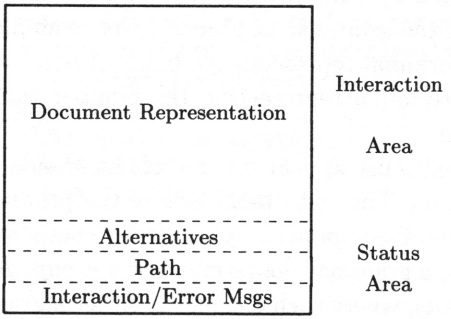

Figure 2: The overall layout of the interactive representation's window.

involves a *family* of editors. One, called the *generalized manipulator*, is used for altering the strict tree portion of the **tnt**. A separate, specialized, editor is provided for each of the classes of user-defined terminals.

3.2.1. The Generalized Manipulator

The generalized manipulator presents the elements of the grammar through *templates*. A template consists of a *button* and an indication of what the template represents. When the button is *selected*, it expands. In the case of a nonterminal, the expansion is another set of buttons indicating the expansion of that nonterminal, enclosed in a box that represents the expanded nonterminal itself. When the element is a user-defined terminal, the expansion presents the specialized editor for that particular type of terminal. System-defined terminals cannot be selected—the system expands them when they are present, generating their values as needed.

The placement of buttons and the placement of the expansion of buttons resembles that the object would take if placed onto a hard copy medium. The intention is to help indicate the semantics of the object by representing its appearance.

When the button corresponding to a user-defined terminal is selected, a window is opened to allow editing of the terminal's contents. Initially, the window corresponds in size and position to the button being expanded. As the information is edited, the window grows temporarily obscuring the display of adjacent objects. When the editing is complete, the information is put into the **tnt** and the **tnt**'s display is adjusted.

The grammar augmentations aid in the generic manipulation of the **tnt**. When a choose all group is displayed, a button is presented for each of its alternatives. When a choose one group is shown, only one of the alternatives is shown. In both cases, the *transform* operation

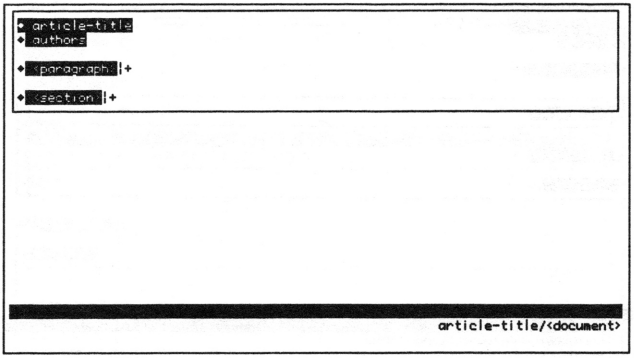

Figure 3: The initial display on creation of a new sectioned document.

allows the element associated with the button to be changed to another alternative from the group.

If a selected element can be replicated (starred or plussed in the grammar) or if it is an element in a group that can be replicated, selecting it causes new buttons to be inserted before and after the expansion, permitting further specification of elements of that type.

The opposite of expanding a button is collapsing a subtree into a button. Selecting this button restores the contents that have been hidden by the collapse. The button's display also includes a brief summary of the collapsed objects' contents.

Figure 2 shows the overall arrangement of the interactive display of the **tnt**. The display is divided into two major areas: the Interaction Area and the Status Area. The Interaction Area contains the templates described above. The Status Area subdivides into three parts. The topmost shows the "alternatives." When a button representing an element from a group is the current selection, this area shows what it could be transformed into. The second part of the Status Area is the "path," which shows the nodes between the root and the current position. The third part shows "interaction and error messages." When the prototype needs additional information from the system's user, it poses its queries in this area. When an error occurs, the explanation of what happened is posted here.

The prototype has been implemented using an 80 column by 24 row terminal. Figure 3 shows the initial display that would result when the prototype was started with the `<document>` as described by the grammar previously shown in Figure 1. The four buttons correspond

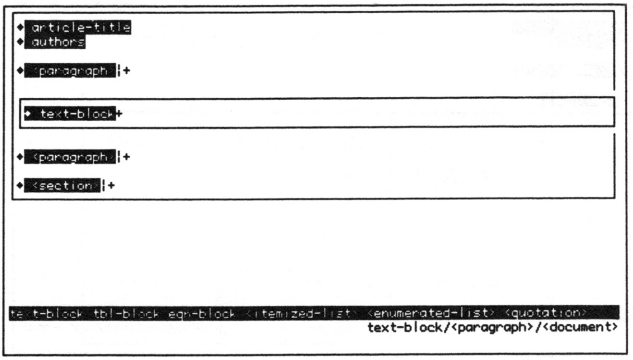

Figure 4: After selection of a **<paragraph>** button.

to the elements on the right hand side of the production. The "|+" following the **<paragraph>** and **<section>** buttons indicates that the elements are in an or list and are plussed. Since **<paragraph>** and **<section>** are in a choose all group, both alternatives are shown.

In Figure 4, the cursor has been moved to the **<paragraph>** button and the button selected. Replacement buttons for new **<paragraph>**s have been inserted before and after the expansion. Since the **text-block** in the **<paragraph>**'s expansion is in a choose one group, only one representative selection is shown. The available alternatives are shown in the status area.

In Figure 5, the **text-block** has been selected, activating the specialized editor for text blocks. Again, replacement buttons have been inserted before and after the expansion.[5]

Figure 6 shows the final state. The text block editor has exited, inserting the contents into the **tnt** and returning control to the generalized manipulator.

3.2.2. Specialized Editors for User-defined Terminals

The specialized editors correspond to the various kinds of **tnt** tree blocks and are invoked by the strict tree's generalized editor. In turn, some of the specialized editors may invoke another instance of the generalized editor to manipulate a strict tree rooted within their data structure.

Three specialized editors are defined in the prototype implementation: a text block editor, a table block editor, and an equation block editor.

[5]The replacement buttons here are known as *ellipsis* buttons and are used because the original element was a member of a choose one group.

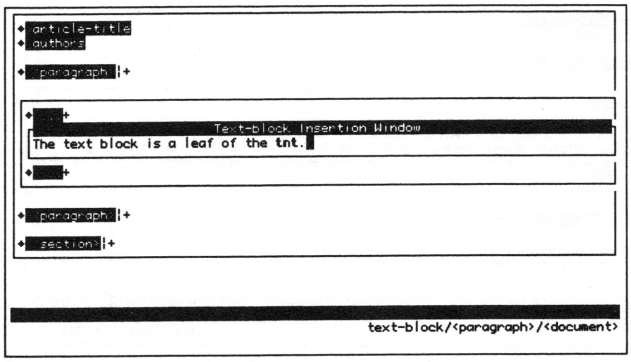

Figure 5: In the specialized editor for text blocks.

The text block editor: The text block editor is perhaps the simplest of the prototype's specialized editors. A text block is modeled as an unbroken string of characters and represents the text of a paragraph or of a substantial part of a paragraph. The text block editor presents this string as a sequence of lines. The editor's operations alter the position of the cursor and insert and delete characters relative to the cursor. The displayed lines are continuously adjusted as characters are added to or deleted from the string. The text block editor also supports font face changes (the provided font faces are roman, italic, bold, and default).

The table block editor: The table block editor is used to create and modify the *structure* of the table and not the *contents*. The table block is modeled as a set of *skeleton nodes* placed into a two-dimensional structure of rows and columns. A skeleton node is in exactly one row and in one column. A table is a set of rectangular *entries*, each of which occupies one or more adjacent skeleton nodes.[6]

The table entries are strict trees, each rooted in a `<tbl-entry>`. Therefore, manipulations on the contents of the table are carried out by invoking a new instance of the generalized manipulator on the entry's tree.

The table block editor provides commands to modify the structure of the table by adding, deleting, and moving rows and columns, commands to create and break up "spans" of skeleton nodes, thereby associating entries with multiple skeleton nodes, and a command that transfers control to the new instance of the generalized manipulator.

[6]A similar model of tables can be found in work by Biggerstaff, Endres, and Forman [BEF84], and indeed in that provided by Unix's **tbl** table processor [Les76].

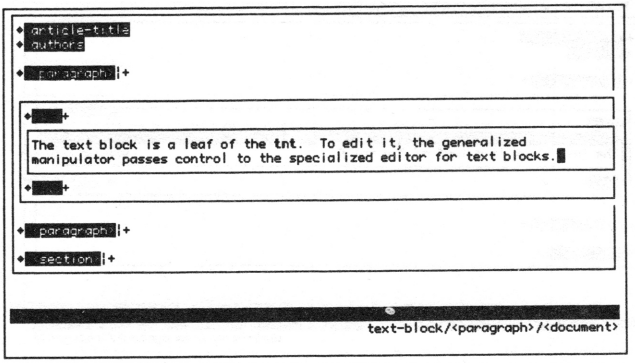

Figure 6: After exiting the text block editor.

The equation block editor: The equation block editor differs from the other editors in that it uses keywords to direct its actions. Input resembles **eqn** or TEX input strings [KC75, Knu84] but the resulting structures are displayed while they are being entered. Characters are placed at the current position as they are entered from the keyboard. The system also provides positioning commands, which are allowed to roam across the partially specified equation changing the current position within the equation.

The equation description is a set of lists of elements. Each element in the list is either a terminal (a character or a special symbol) or an *entry record*. Entry records point to additional lists related to each other by the physical grid of Figure 7.

An empty list is displayed as an isolated button (without any attached identifiers). Hence in the equation domain, buttons indicate the places where it is possible to place material.

As characters are entered, keywords are detected and acted on. Keywords fall into two main classifications. The first are those that represent special symbols not present on the keyboard. For example, the sequence "**sum**" causes a large summation sign to replace the string on the display and in the data structure. "**+-**" or "**-+**" produce the "$\pm$" symbol, and similar key mappings produce "$\geq$", and "$\leq$".

The second class of keywords are those that modify the data structure, generally creating new lists that can then be inserted into. The **over** keyword inserts an "over" node (for a fraction) into the equation list. The "over" node has two empty lists, representing the numerator and the denominator of the fraction separated by horizontal bar the

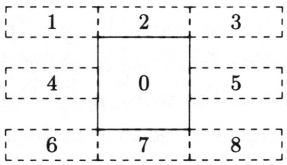

Figure 7: The grid that constrains equation element placement.

width of which is the same as the width of the widest of the two lists. **sup** and **sub** take the preceding token and convert it into a "superscript" or "subscript" node that contains an empty list representing the superscript or subscript.

The equation block editor is largely table driven. In particular, the keywords are defined in a table and the translations and graphical representations for the special symbol keywords are defined solely by table.

3.3. Region-oriented Manipulation

A goal of this research is to allow concrete-oriented manipulations of the document and the region commands are one of the means provided for this purpose. Region commands are carried out on an arbitrary contiguous region of the display. The selection of the region on which the command will operate can be made using a combination of cursor-oriented selections and tree-oriented selections.

In cursor-oriented selection, the cursor is initially positioned at one end or the other of the selection and cursor motions cause the selection to expand or contract. With tree-oriented selection, the selection is increased to encompass the parent of a node.

A postfix command notation is used: the region selection is made first and then the command is specified. Region commands provide the means to copy, delete, and move parts of the document, as well as to set the region's font.

A region may span across different structure contexts. When this occurs, the system converts the operation into a sequence of operations each within a single context through the use of heuristics. After the actions have been carried out, a routine that analyzes the tree based on the grammar is called to fix up the tree and the display.

4. Conclusions

An important pedagogical distinction within this work has been the distinction of the Exact-representation Editor/Formatters from the Integrated Editor/Formatters. The focus in this work has been to examine Integrated Editor/Formatters that are not also Exact-representation

Editor/Formatters. One lesson learned from TEX is that high-quality document formatting benefits from use of algorithms that may be quite expensive computationally in the worst case. Developing an Integrated but not Exact-representation Editor/Formatter allows the presentation of information in a form that can be quickly recognized by its appearance while using algorithms more suited to interactive manipulation.

Also important has been the separation of the document model into a single internal representation and multiple external representations. Scribe alters the appearance of the objects in a document to best suit the intended output device. In essence, this use of multiple external representations is a generalization of that concept.

The separation of the **tnt** into the strict tree and the tree block objects allows definition of different document types while providing a consistent means for their manipulation. The basic building blocks (the tree blocks) are the same across the different document types. Generic operations on the strict tree allow manipulation of the document type's objects. The tree/non-tree/tree/... relationships in the **tnt** extend this consistency.

Interestingly, the framework that allows the multiple editors and tree blocks to co-exist makes it relatively easy to add new block object types to the **tnt** and to change the definition of the existing block object types. As such, it resembles the Unix pipe concept, which allows the easy interconnection of separate cooperating programs.

The region-oriented commands demonstrate that it is possible to allow concrete-like manipulations of a document represented as abstract, hierarchically structured objects without requiring that the system's user manipulate a hidden data structure.

The prototype is far from being a production system and its display, limited by the standard terminal, is quite coarse. It has, however, served as a useful testbed for investigation and implementation of the algorithms that allow the system to generate a display and allow document manipulation based on a grammatical description. A project of particular interest will be to translate the techniques developed here into the environment provided by bitmapped displays with attached pointing devices.

References

[App84] *MacWrite*. Apple Computer, Inc., 1984.

[BEF84] Ted J. Biggerstaff, D. Mac Endres, and Ira R. Forman. TABLE: Object oriented editing of complex structures. In *Proceedings—International Conference on Software Engineering*, pages 334–345, IEEE

Computer Society, 1984.

[BP80] S. A. Borkin and J. M. Prager. *Some issues in the design of an editor-formatter for structured documents.* Technical Report, IBM Cambridge Scientific Center, November 1980.

[FSS82] Richard Furuta, Jeffrey Scofield, and Alan Shaw. Document formatting systems: Survey, concepts, and issues. *Computing Surveys*, 14(3):417–472, September 1982.

[Fur86] Richard Furuta. *An Integrated, but not Exact-Representation, Editor/Formatter.* Ph.D. dissertation, University of Washington, Department of Computer Science, Seattle, WA, 1986. In preparation.

[KC75] Brian W. Kernighan and Lorinda L. Cherry. A system for typesetting mathematics. *Communications of the ACM*, 18(3):151–157, March 1975.

[Kim84] Gary Dean Kimura. *A Structure Editor and Model for Abstract Document Objects.* Ph.D. dissertation, University of Washington, Department of Computer Science, Seattle, WA, 1984.

[Knu84] Donald E. Knuth. *The TEXbook.* Addison-Wesley, 1984.

[KS84] Gary D. Kimura and Alan C. Shaw. The structure of abstract document objects. *Proceedings of the Second ACM-SIGOA Conference on Office Information Systems, SIGOA Newsletter*, 5(1–2):161–169, June 1984.

[Lam78] Butler W. Lampson. Bravo manual. In B. W. Lampson and E. A. Taft, editors, *Alto User's Handbook*, Computer Science Laboratory, Xerox Palo Alto Research Center, November 1978.

[Lam85] Leslie Lamport. *LATEX: A Document Preparation System.* Addison-Wesley, 1985.

[Les76] M. E. Lesk. *Tbl—A program to format tables.* Computer Science Technical Report 49, Bell Laboratories, Murray Hill, N.J., September 1976.

[PJN85] Arno J. H. M. Peels, Norbert J. M. Janssen, and Wop Nawijn. Document architecture and text formatting. *ACM Transactions on Office Information Systems*, 3(4):347–369, October 1985.

[QV86] Vincent Quint and Iréne Vatton. GRIF: An interactive system for structured document manipulation. In *Proceedings of the International Conference on Text Processing and Document Manipulation*, April 1986. This volume.

[Rei80] Brian K. Reid. *Scribe: A Document Specification Language and its Compiler.* Ph.D. dissertation, Carnegie-Mellon University Computer Science Department, Pittsburgh, PA, October 1980.

[Sha80] Alan C. Shaw. *A Model for Document Preparation Systems.* Technical Report 80-04-02, University of Washington, Department of Computer Science, Seattle, WA, April 1980.

[SIKV82] David Canfield Smith, Charles Irby, Ralph Kimball, and Bill Verplank. Designing the Star user interface. *Byte*, 7(4):242–282, April 1982.

[Uni84] Unilogic, Ltd. *Scribe Introductory User's Manual.* Unilogic, Ltd., Pittsburgh, fourth edition, April 1984.